PEP 3131 – Supporting Non-ASCII Identifiers

peps.python.org/pep-3131/

Abstract

This PEP suggests to support non-ASCII letters (such as accented characters, Cyrillic, Greek, Kanji, etc.) in Python identifiers.

Rationale

Python code is written by many people in the world who are not familiar with the English language, or even well-acquainted with the Latin writing system. Such developers often desire to define classes and functions with names in their native languages, rather than having to come up with an (often incorrect) English translation of the concept they want to name. By using identifiers in their native language, code clarity and maintainability of the code among speakers of that language improves.

For some languages, common transliteration systems exist (in particular, for the Latin-based writing systems). For other languages, users have larger difficulties to use Latin to write their native words.

Common Objections

Some objections are often raised against proposals similar to this one.

People claim that they will not be able to use a library if to do so they have to use characters they cannot type on their keyboards. However, it is the choice of the designer of the library to decide on various constraints for using the library: people may not be able to use the library because they cannot get physical access to the source code (because it is not published), or because licensing prohibits usage, or because the documentation is in a language they cannot understand. A developer wishing to make a library widely available needs to make a number of explicit choices (such as publication, licensing, language of documentation, and language of identifiers). It should always be the choice of the author to make these decisions - not the choice of the language designers.

In particular, projects wishing to have wide usage probably might want to establish a policy that all identifiers, comments, and documentation is written in English (see the GNU coding style guide for an example of such a policy). Restricting the language to ASCII-only identifiers does not enforce comments and documentation to be English, or the identifiers actually to be English words, so an additional policy is necessary, anyway.

Specification of Language Changes

The syntax of identifiers in Python will be based on the Unicode standard annex UAX-31, with elaboration and changes as defined below.

Within the ASCII range (U+0001..U+007F), the valid characters for identifiers are the same as in Python 2.5. This specification only introduces additional characters from outside the ASCII range. For other characters, the classification uses the version of the Unicode Character Database as included in the `unicodedata` module.

The identifier syntax is `<XID_Start> <XID_Continue>*`.

The exact specification of what characters have the XID_Start or XID_Continue properties can be found in the DerivedCoreProperties file of the Unicode data in use by Python (4.1 at the time this PEP was written). For reference, the construction rules for these sets are given below. The XID_* properties are derived from ID_Start/ID_Continue, which are derived themselves.

`ID_Start` is defined as all characters having one of the general categories uppercase letters (Lu), lowercase letters (Ll), titlecase letters (Lt), modifier letters (Lm), other letters (Lo), letter numbers (Nl), the underscore, and characters carrying the Other_ID_Start property. `XID_Start` then closes this set under normalization, by removing all characters whose NFKC normalization is not of the form ID_Start ID_Continue* anymore.

`ID_Continue` is defined as all characters in `ID_Start`, plus nonspacing marks (Mn), spacing combining marks (Mc), decimal number (Nd), connector punctuations (Pc), and characters carrying the Other_ID_Continue property. Again, `XID_Continue` closes this set under NFKC-normalization; it also adds U+00B7 to support Catalan.

All identifiers are converted into the normal form NFKC while parsing; comparison of identifiers is based on NFKC.

A non-normative HTML file listing all valid identifier characters for Unicode 4.1 can be found at https://web.archive.org/web/20081016132748/http://www.dcl.hpi.uni-potsdam.de/home/loewis/table-3131.html.

Policy Specification

As an addition to the Python Coding style, the following policy is prescribed: All identifiers in the Python standard library MUST use ASCII-only identifiers, and SHOULD use English words wherever feasible (in many cases, abbreviations and technical terms are used which aren't English). In addition, string literals and comments must also be in ASCII. The only exceptions are (a) test cases testing the non-ASCII features, and (b) names of authors. Authors whose names are not based on the Latin alphabet MUST provide a Latin transliteration of their names.

As an option, this specification can be applied to Python 2.x. In that case, ASCII-only identifiers would continue to be represented as byte string objects in namespace dictionaries; identifiers with non-ASCII characters would be represented as Unicode

strings.

Implementation

The following changes will need to be made to the parser:

1. If a non-ASCII character is found in the UTF-8 representation of the source code, a forward scan is made to find the first ASCII non-identifier character (e.g. a space or punctuation character)
2. The entire UTF-8 string is passed to a function to normalize the string to NFKC, and then verify that it follows the identifier syntax. No such callout is made for pure-ASCII identifiers, which continue to be parsed the way they are today. The Unicode database must start including the Other_ID_{Start|Continue} property.
3. If this specification is implemented for 2.x, reflective libraries (such as pydoc) must be verified to continue to work when Unicode strings appear in `__dict__` slots as keys.

Open Issues

John Nagle suggested consideration of Unicode Technical Standard #39, which discusses security mechanisms for Unicode identifiers. It's not clear how that can precisely apply to this PEP; possible consequences are

- warn about characters listed as "restricted" in xidmodifications.txt
- warn about identifiers using mixed scripts
- somehow perform Confusable Detection

In the latter two approaches, it's not clear how precisely the algorithm should work. For mixed scripts, certain kinds of mixing should probably allowed - are these the "Common" and "Inherited" scripts mentioned in section 5? For Confusable Detection, it seems one needs two identifiers to compare them for confusion - is it possible to somehow apply it to a single identifier only, and warn?

In follow-up discussion, it turns out that John Nagle actually meant to suggest UTR#36, level "Highly Restrictive".

Several people suggested to allow and ignore formatting control characters (general category Cf), as is done in Java, JavaScript, and C#. It's not clear whether this would improve things (it might for RTL languages); if there is a need, these can be added later.

Some people would like to see an option on selecting support for this PEP at run-time; opinions vary on what precisely that option should be, and what precisely its default value should be. Guido van Rossum commented that a global flag passed to the interpreter is not acceptable, as it would apply to all modules.

Discussion

Ka-Ping Yee summarizes discussion and further objection as such:

1. Should identifiers be allowed to contain any Unicode letter?
   Drawbacks of allowing non-ASCII identifiers wholesale:

   1. Python will lose the ability to make a reliable round trip to a human-readable display on screen or on paper.
   2. Python will become vulnerable to a new class of security exploits; code and submitted patches will be much harder to inspect.
   3. Humans will no longer be able to validate Python syntax.
   4. Unicode is young; its problems are not yet well understood and solved; tool support is weak.
   5. Languages with non-ASCII identifiers use different character sets and normalization schemes; PEP 3131's choices are non-obvious.
   6. The Unicode bidi algorithm yields an extremely confusing display order for RTL text when digits or operators are nearby.
2. Should the default behaviour accept only ASCII identifiers, or should it accept identifiers containing non-ASCII characters?
   Arguments for ASCII only by default:

   1. Non-ASCII identifiers by default makes common practice/assumptions subtly/unknowingly wrong; rarely wrong is worse than obviously wrong.
   2. Better to raise a warning than to fail silently when encountering a probably unexpected situation.
   3. All of current usage is ASCII-only; the vast majority of future usage will be ASCII-only.
   3. It is the pockets of Unicode adoption that are parochial, not the ASCII advocates.
   4. Python should audit for ASCII-only identifiers for the same reasons that it audits for tab-space consistency
   5. Incremental change is safer.
   6. An ASCII-only default favors open-source development and sharing of source code.
   7. Existing projects won't have to waste any brainpower worrying about the implications of Unicode identifiers.
3. Should non-ASCII identifiers be optional?
   Various voices in support of a flag (although there's been debate over which should be the default, no one seems to be saying that there shouldn't be an off switch)

4. Should the identifier character set be configurable?
   Various voices proposing and supporting a selectable character set, so that users can get all the benefits of using their own language without the drawbacks of confusable/unfamiliar characters

5. Which identifier characters should be allowed?
    1. What to do about bidi format control characters?
    2. What about other ID_Continue characters? What about characters that look like punctuation? What about other recommendations in UTS #39? What about mixed-script identifiers?
6. Which normalization form should be used, NFC or NFKC?
7. Should source code be required to be in normalized form?